Implementação do AES-128 em VHDL

Sistemas Digitais

Jiovana Sousa Gomes(gomesjiovana@gmail.com)

Engenharia de Computação

Bagé, 2019

Introdução

Este é um relatório mostrando os principais resultados da implementação de uma arquitetura em VHDL para execução do algoritmo de criptografia AES de 128 bits, segundo as especificações do trabalho. Foi implementada a parte de encriptação do algoritmo, obtendo, a partir de uma entrada e uma chave, em hexadecimal de 32 dígitos, um texto cifrado de mesmo tamanho.

O algoritmo - Encriptação

É composto por uma série de operações sobre a entrada e a chave, sendo dividido em rounds. Primeiramente, deve-se considerar a entrada como uma matriz 4X4, pois as operações são feitas sob as linhas e colunas da mesma. Essas operações são:

- **SubBytes**: substitui os 16 bytes da entrada por outros mapeados a partir de uma tabela 16x16 conhecida como SBOX;
- **ShiftRows**: rotaciona para esquerda as linhas da entrada, de acordo com o número da linha, ou seja, a linha 0 não rotaciona, a linha 1 rotaciona uma posição, a linha 2 rotaciona duas posições e a linha 3 rotaciona três posições;
- **MixColumns**: etapa que envolve uma série de cálculos de multiplicação polinomial entre as colunas da entrada e uma outra matriz chamada de 'A', que apresenta valores constantes;
- **AddRoundKey**: realiza uma xor entre a entrada e a chave de round.

Além disso, a cada round, existe uma chave de round, que é obtida através de um processo separado chamado de 'Key Schedule', que gera uma chave diferente para cada um dos rounds. Para o round 0, essa chave é a chave de entrada. Após isso, é usada a chave do round anterior para se obter a atual.

O key Schedule realiza operações semelhantes às do algoritmo principal, fazendo substituições com a SBOX, rotações e multiplicações.

A execução do AES dá-se da seguinte forma:

- No round 0, realiza apenas uma AddRoundKey da entrada e chave inicial.
- Para os rounds 1-9, são realizadas em sequência as 4 operações descritas acima.
- No round 10, pula-se a etapa MixColumns.
- A saída do round 10 representa o texto cifrado.

A imagem 1 ilustra a execução do algoritmo, demonstrando o loop dos rounds 1-9. A variável ‘i’ representa o número do round. O vetor W refere-se às chaves de round. Nr seria ‘número de rounds’. Pois existem diferentes implementações do AES, onde o número de rounds pode ser mais que 10, dependendo do tamanho da chave.

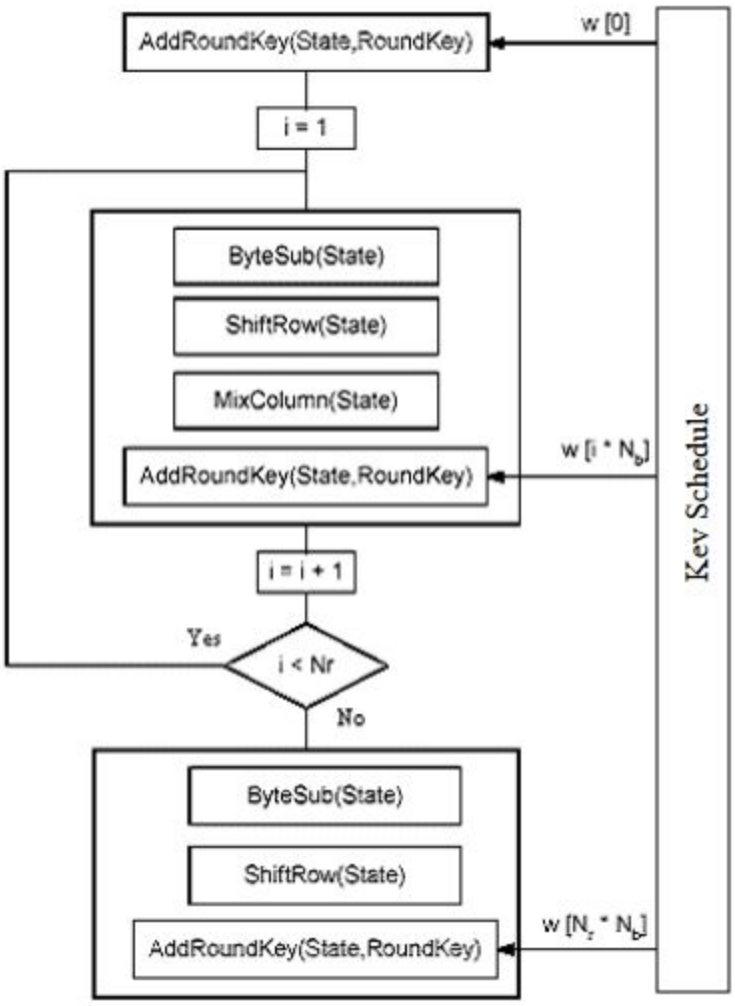

Figura 1: Representação genérica da fase de encriptação do AES.

Implementação em VHDL

Os requisitos do trabalho pediam uma implementação sequencial, o que implica em realizar as etapas do AES de forma a salvar suas saídas em registradores, de outra forma, o trabalho poderia ser realizado de forma puramente combinacional, realizando tudo de uma única vez mas isso implicaria em uma execução muito lenta do algoritmo, não usando o potencial que a linguagem tem a oferecer também.

Inicialmente foi feita uma pesquisa para se entender o funcionamento de cada uma das etapas do algoritmo, e então foi encontrado um trabalho em VHDL que implementou essas etapas sob a forma de funções, com cada função fazendo as operações de cada etapa. Esse trabalho é o primeiro nas referências. Foi usada essa implementação como base para realização deste trabalho, porém não sob a forma de funções, pois o conteúdo destas é executado de forma sequencial na linguagem, por isso achou-se melhor fazer de forma paralela deixando as operações direto na arquitetura dos componentes.

Assim, a implementação foi pensada na forma de existirem componentes para cada etapa do algoritmo, e entre esses componentes, foram colocados registradores para guardar a saída de cada etapa. Também foi necessária a existência de alguns muxes e um contador para contar os rounds. A SBOX foi implementada sob a forma de uma ROM com 2 endereços e 2 saídas. Optou-se por replicar a ROM algumas vezes para se tentar obter acessos simultâneos de todos os bytes no mesmo ciclo, de forma que a operação de subBytes fosse realizada toda em um único ciclo.

Sobre a máquina de estados, ela ficou com um total de 8 ciclos para a execução completa do algoritmo. A entrada e a chave inicial são constantes dentro do código, pois o device FPGA usado possui uma restrição de pinos que impede que essas informações sejam passadas como entrada, de forma que apenas o clock e o reset são entradas, e o texto cifrado fica sendo a saída, além dos sinais de controle.

O trabalho foi implementado usando os softwares Quartus II e ModelSim.

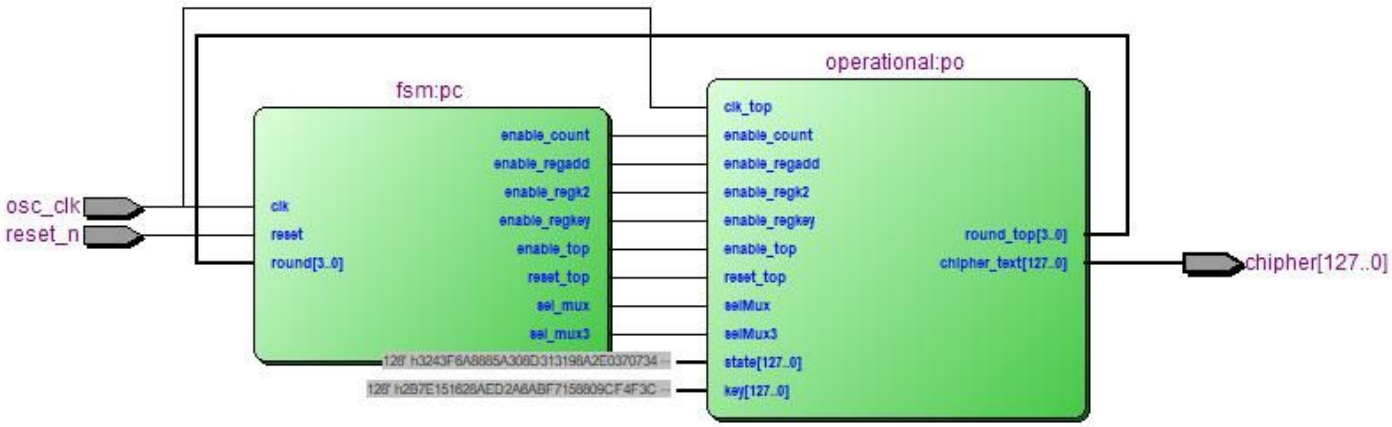

Figura 2: Visão de topo da arquitetura implementada

A imagem 2 apresenta o topo da arquitetura, mostrando que existem dois componentes principais: a parte operativa onde o algoritmo é executado, e a parte de controle, onde está a máquina de estados. É possível visualizar os sinais de controle saindo da parte de controle e entrando na parte operativa. O contador está localizado na parte operativa e é usado como entrada para a máquina de estados, tendo o nome de ‘round’.

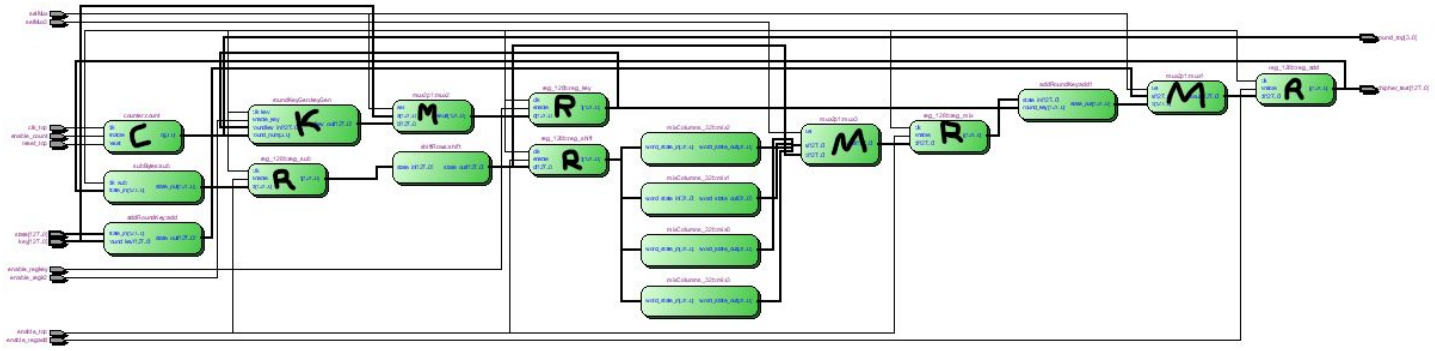

Figura 3: Componentes internos da parte operativa

A imagem 3 está em tamanho reduzido demais para poder ser interpretada de maneira adequada, mas ela mostra os componentes internos que estão contidos na parte operativa. Os componentes com ‘R’ representam registradores. Aqueles com ‘M’ são muxes. O componente com ‘K’ é o Key Schedule e o com ‘C’ é o contador. Os demais componentes são os módulos do algoritmo, sendo o do meio, que apresenta 4 partes, o MixColumns, pois esse foi dividido em 4 partes de 32 bits para simplificação do código e execução mais rápida.

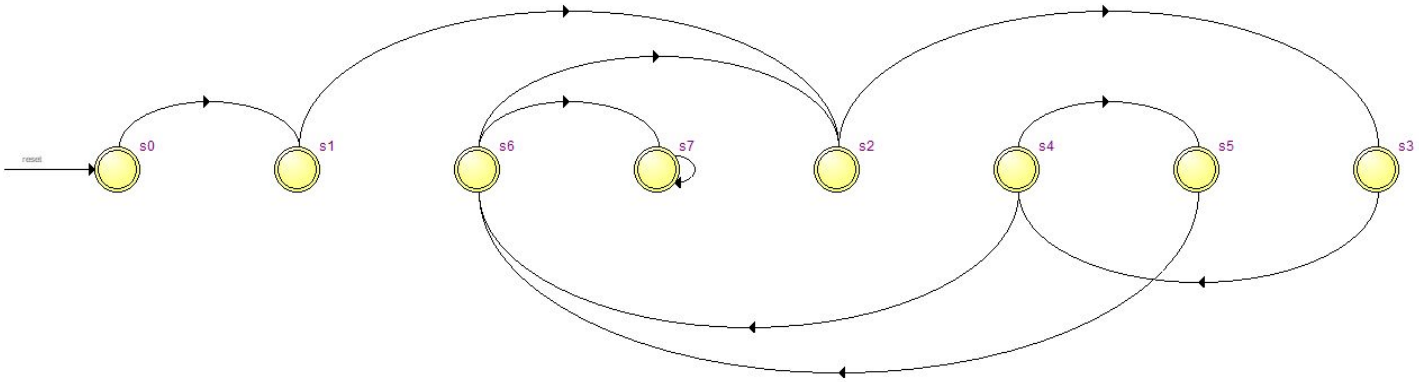

Figura 4 : representação gráfica da máquina de estados implementada.

A máquina de estados pode ser interpretada como uma FSM de Mealy, pois o valor do round é usado para definir alguns sinais de controle, principalmente no round 10, para fazer pular o MixColumns. Com exceção do round 1 e do 10, os demais executam em 5 estados consecutivos. O 1 executa com um estado a menos pois uma das operações é feita antes, no round 0 (o addRoundKey inicial). O 10 executa com um a menos pois ele evita o Mix. Ou seja, cada round tem em média 5 ciclos de execução, pois cada estado dura 1 ciclo de clock.

Resultados

A seguir constam os resultados de compilação da arquitetura no Quartus, de acordo com as especificações do trabalho para o device EP3C25F324C6.

Frequência máxima (opção slow 1200mV 85C) - 206.06 MHz.
Potência dissipada total - 203.31 mW (62.51 mW dinâmico, 81.31 mW estático e 59.49 mW de I/O).
Área ocupada - 967 elementos lógicos (4% do total), 812 registradores, 130 pinos (60% do total) e 20480 bits de memória (3% do total).

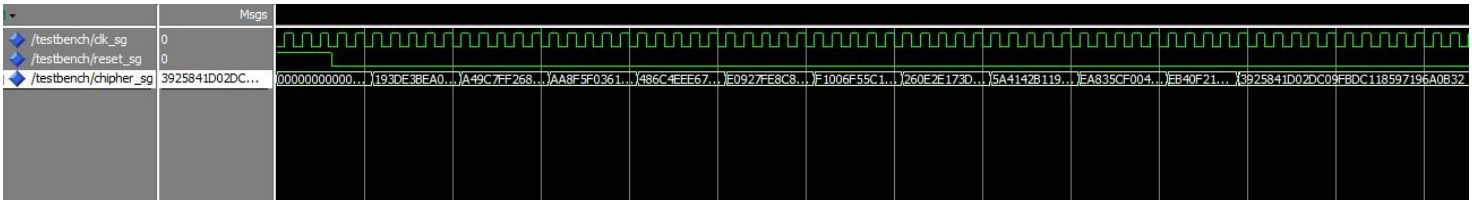

Figura 5 : simulação funcional da execução da arquitetura, no ModelSim.

A execução completa do algoritmo, para um bloco de 128 bits, levou 273 ns, segundo a execução funcional no ModelSim e no Quartus. Executando simulação temporal no Quartus, para um clock de 5 ns, obteve-se o resultado após aproximadamente 283 ns. O reset, definido no testbench, está ativo até 16 ns.

Parâmetros de entrada:

State = 3243f6a8885a308d313198a2e0370734
Key = 2b7e151628aed2a6abf7158809cf4f3c

Saída esperada:

Chipher = 3925841d02dc09fbdc118597196a0b32

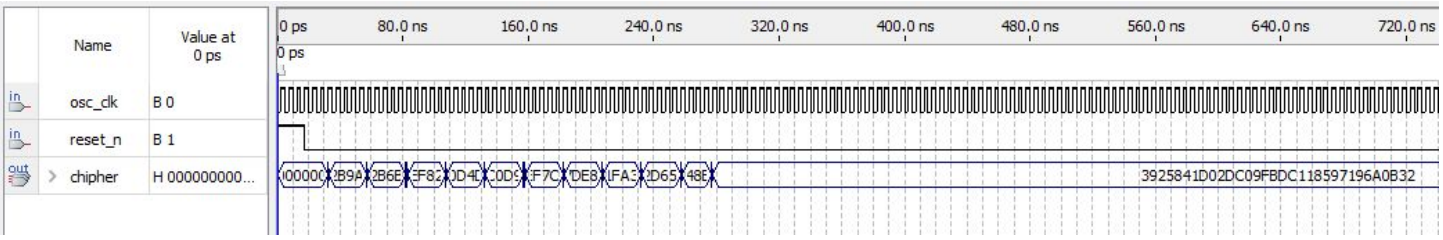

Figura 6: Simulação temporal no Quartus

O projeto anteriormente havia sido executado sem contabilizar atrasos de I/O. Após adicionar esses atrasos no arquivo .sdc, a frequência reduziu , porém o projeto tem valores mais realistas.

O caminho crítico identificado antes dos atrasos, pelo analisador de tempos, foi o do componente gerador das chaves de round. Tentou-se eliminar esse caminho crítico com a adição de um registrador dentro desse componente, para guardar um resultado parcial dos cálculos. Após adicionar os atrasos, o caminho crítico apontado é a agora o componente SubBytes. Isso faz sentido em razão dele ser onde estão as ROMs paralelas.

Conclusões

A implementação desse algoritmo foi uma tarefa que parecia inicialmente complicada, pois criptografia envolve muita matemática, e entender cada etapa levou certo tempo. Porém, ao decorrer do processo, a implementação mostrou-se simples, sem muitos problemas.

Esse projeto pode ainda ser otimizado, ao tentar-se fazer uma serialização da entrada de forma a poder passar a entrada e a chave como entradas para a arquitetura sem ter o problema do número de pinos. Isso aumentaria o número de ciclos da execução e adicionaria alguns estados a mais, mas evitaria a necessidade de ficar mexendo no código para poder alterar esses parâmetros.

Também, deve-se analisar mais formas de otimização da arquitetura e talvez outras formas de implementar, para conseguir vencer os caminhos críticos e conseguir obter uma frequência de operação ainda maior.

O código fonte do projeto encontra-se disponível em https://github.com/Jiovana/AES-128, com projetos do Quartus e do ModelSim, ambos com testbenches para execução.

Referências e inspirações

- https://pdfs.semanticscholar.org/d994/33c3e9b75a111bbe6700e76dab608cf70448.pdf, acesso entre set e nov 2019.
- https://pt.stackoverflow.com/questions/43492/como-funciona-o-algoritmo-de-criptografia-aes, acesso entre set e nov 2019.
- https://www.samiam.org/key-schedule.html, acesso entre set e nov 2019.
- E wikipedia.